

PATENT

Attorney Docket No.: 042390P13414

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

TITLE OF THE INVENTION

**METHOD AND APPARATUS TO
EXECUTE INSTRUCTIONS IN A PROCESSOR**

INVENTORS

**ERIC SPRANGLE
MICHAEL J. HAERTEL
DAVID J. SAGER**

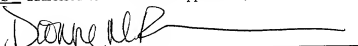
Prepared by

**BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CA 90025-1026
(503) 684-6200**

Express Mail Certificate Under 37 CFR 1.10

This paper and any papers indicated as being transmitted herewith, are being deposited with the U.S. Postal Service on this date January 2, 2002, in an Express Mail envelope, as Express Mail Number EL 863955417 US addressed to Box Patent Application, Commissioner For Patents, Washington, D.C. 20231.

01/02/02
Date


Signature

COPYRIGHT NOTICE

[0001] Contained herein is material that is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction of the patent disclosure by any person as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all rights to the copyright whatsoever.

BACKGROUND OF THE INVENTION

Field of the Invention

[0002] The present invention is related to the field of electronics. In particular, the present invention is related to a method and apparatus to execute instructions in a processor.

Description of the Related Art

[0003] Out-of-order processors commonly use a pipelining technique wherein multiple instructions are overlapped in execution in an effort to improve the overall performance of the processor e.g., a microprocessor. Pipelining is a technique that exploits parallelism among the processor instructions in a sequential instruction stream. Pipelining increases the processor's instruction throughput, without reducing the execution time of an individual instruction. This allows for a processor to execute a program faster with a lower total execution time, even though no single instruction runs faster.

[0004] In a pipelined processor, the latency from scheduling an instruction to executing the instruction, and then confirming the instruction executed correctly may be significantly longer than the latency of the instruction. Therefore, to minimize the effective latency of the instruction, dependent instructions should be scheduled before confirming that the first instruction executed correctly. In a pipelined processor, a scheduler speculatively schedules instructions assuming that all instructions will execute properly (e.g., all load instructions will hit in data cache). Thus, a situation may arise that prevents the next instruction in the instruction stream from executing correctly during its designated clock cycle if the instruction requires the results of the previous instruction in order for it to execute correctly. When this occurs, the processor must determine which instructions have not executed properly, either because the instruction was not able to generate the correct result given the correct input values, or the instruction could not generate the correct result because it was not provided with the correct input values.

BRIEF SUMMARY OF THE DRAWINGS

[0005] Examples of the present invention are illustrated in the accompanying drawings. The accompanying drawings, however, do not limit the scope of the present invention. Similar references in the drawings indicate similar elements.

[0006] Figure 1 illustrates a block diagram of a processor according to one embodiment of the invention;

[0007] Figure 2 illustrates cache memory with validity bits according to one embodiment of the invention;

[0008] Figure 3 illustrates a flow diagram for processing instructions according to one embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

[0009] Described is a method and apparatus to process instructions using a validity bit. In the following description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one of ordinary skill in the art that the present invention may be practiced without these specific details. In other instances, well-known architectures, steps, and techniques have not been shown to avoid unnecessarily obscuring the present invention.

[0010] Parts of the description is presented using terminology commonly employed by those skilled in the art to convey the substance of their work to others skilled in the art. Also, parts of the description will be presented in terms of operations performed through the execution of programming instructions. As well understood by those skilled in the art, these operations often take the form of electrical, magnetic, or optical signals capable of being stored, transferred, combined, and otherwise manipulated through, for instance, electrical components.

[0011] **Figure 1** illustrates a block diagram of a processor according to one embodiment of the invention. As illustrated in Figure 1, computer system 100 comprises a processor 77 that is coupled to various components of computer system 100, e.g., a memory unit (not shown) via a system bus 66. The memory unit may include random access memory, read only memory or some other permanent or temporary storage device. In one embodiment, processor 77 is an out-of-order processor.

[0012] Processor 77 includes a scheduler 105 that receives instructions (e.g., from an instruction queue) via bus 150. The instructions received by processor 77 are micro-operations (i.e., instructions generated by transforming complex instructions into fixed

length instructions). Each micro-operation or instruction has one or more sources (from which data is read) and at least one destination (to which data is written). In one embodiment, the source or the destination may be one or more registers within processor 77, cache memory, or even permanent and/or temporary memory (e.g., random access memory RAM).

[0013] Scheduler 105 is coupled to an execution unit 115. In one embodiment, scheduler 105 sends instructions from the instruction queue, or instructions from checker unit 145 to execution unit 115 for execution. Execution unit 115 executes instructions received from scheduler 105. Execution unit 115 may be a floating-point arithmetic logic unit (ALU), a load executing unit (i.e., an executing unit that computes the address location of data, and loads the data from the computed address location), etc.

[0014] Executing unit 115 is coupled to one or more registers 120A, 120B, ...120N. Although, in the embodiment of Figure 1, only three registers (i.e., 120A, 120B, and 120N) are illustrated, other embodiments may have more than three registers as illustrated by the dashed line in between registers 120A and 120B in Figure 1. In one embodiment, the registers are general-purpose registers and data may be read from and written to each of the registers. In one embodiment, each register has a bit (called the validity bit) stored in register locations 125A-N in corresponding registers 120A-N that determines the validity of the data in each register. Thus, each register may have an additional bit (i.e., a validity bit) that is contiguous with the data bits in the register. In some embodiments every register has a validity bit to determine the validity of the data in the register, in alternate embodiments, some registers may have a validity bit, and other registers may not. In one embodiment, the validity bit is not contiguous with the data bits in the registers but is maintained separate from the register (e.g., in a table).

However, a one to one correspondence is maintained between the data in each register and the validity bit. In one embodiment, if the data in a particular register is invalid data, then the validity bit may be set to a logic '1', else the validity bit it is set to a logic '0'.

[0015] In one embodiment, the validity bit may be set to a logic '1' if a cache 'hit' occurs, else if a cache 'miss' occurs the validity bit is set to a logic '0'. A cache miss occurs, for example, if the address tag of the cache block that contains the desired information does not match the block address from the processor.

[0016] **Figure 2** illustrates cache with validity bits according to one embodiment of the invention. In Figure 2, a set associative cache 200 is illustrated, with set 0 having two blocks 0 and 1. Data is stored in the cache in rows 215A-215N. For a given block of cache, e.g., block 0, each row of data 215A-215N in a block has a corresponding validity bit (e.g., validity bits 205A-205N). So also for block 1, rows 215A-215N have corresponding validity bits 210A-210N. The validity bits are set to a logic '0' when a cache hit occurs, else it is set to a logic '1' (i.e., when a cache miss occurs).

[0017] In one embodiment, as illustrated in Figure 1, when an instruction (e.g., a load instruction) is scheduled for execution by scheduler 105, the data including the validity bit are read from cache into registers in processor 77 in accordance with the load instruction. Thus, the registers within processor 77 receive not only the data from cache, but also the validity bit. In particular, the register sizes are increased by at least one bit to accommodate both the data and the validity bit. Although the example above illustrates setting the validity bit whenever a cache miss occurs, other embodiments set the validity bit whenever the data in a register, or a memory location, is invalid. For example, if during execution of a shift operation the next instruction scheduled by the scheduler for execution is executed by execution unit 115, and the next instruction

requires the results from the shift operation, then the register that is used as the source register for the next instruction has its validity bit set to indicate invalid data in the register because the result of the shift operation is not yet available.

[0018] In one embodiment, a data validity circuit 135 (e.g., an AND gate) is coupled to the source registers in processor 77. The data validity circuit determines the validity of the data in the source, e.g., in source registers and indicates the validity of the data in a destination (e.g., a destination register) as follows: If any source register has invalid data (e.g., the validity bit is a logic '0') then the output of the data validity circuit is logic '0', i.e., the data validity circuit 135 sets the validity bit of the destination (e.g., a destination register) to a logic '0'.

[0019] In one embodiment, checker unit 145 has its inputs coupled to execution unit 115, and to one or more destination registers. In particular, checker unit 145 has its input coupled to one or more register locations that store the validity bits (e.g., the validity bit stored in register location 125N of register 120N). In addition, checker unit 145 has its outputs e.g., outputs 116 and 161 coupled to scheduler 105. In one embodiment, the instruction from execution unit 115, and the validity bit corresponding to the output registers are input into checker unit 145. If the validity bit in register location 125N of register 120N indicates that the data in register 120N is invalid, then checker unit 145 re-schedules for execution the instruction that caused the data to be invalid. In order to re-schedule the instruction for execution, checker unit 145 may activate a scheduler control signal 116 that causes scheduler 105 to stop scheduling instructions for the execution unit 115. In one embodiment, scheduler control signal 116 is eliminated as scheduler 105 automatically schedules the oldest ready instructions for scheduling (i.e., instructions that are old in time and are ready for scheduling). In one embodiment, checker unit 145 outputs the instruction that caused the data to be

invalid for re-execution via re-schedule line 161. In alternate embodiments, the instruction that needs to be re-executed is stored within scheduler 105 and is automatically scheduled for re-execution, as it is the oldest ready instruction. In one embodiment, checker unit 145 has one or more buffers to store instructions from execution unit 115 along with the corresponding validity bit. Checker unit 145 re-schedules for execution those instructions that have a corresponding validity bit that indicates invalid data, and retires those instructions that execute properly (i.e., with valid data in the output register).

[0020] Although the checker unit 145 in the embodiment of Figure 1 is illustrated as a separate unit, other embodiments may have the checker unit incorporated within execution unit 115, or within scheduler 105 to form an integral unit. In one embodiment, the checker unit 145 may be eliminated from processor 77. The validity bit may be used as a control signal to control the operation of scheduler 105. For example, when the data in a destination register is invalid, the validity bit is used to stop the scheduler from sending new instructions to the execution unit, and the instruction that caused the invalid data to be generated is re-execution by execution unit 115.

[0021] If during the time an instruction is re-scheduled for execution the data in the source registers become valid, then following the re-execution of the instruction the validity bit in output register 120N is set to indicate valid data. In particular, each time the instruction is re-executed, the validity bit is re-generated, so if the data in the source becomes valid, the result in the destination is valid and the validity bit in the destination indicates a valid result.

[0022] As illustrated in Figure 1 retirement unit 155 is coupled to checker unit 145. Retirement unit receives instructions from checker unit 145 that have properly executed

by execution unit 115. Retiring instructions free up processor resources and permits additional instructions to execute.

[0023] Thus, as illustrated in Figure 1, if either, or both, source registers 120A or 120B have invalid data (indicated by validity bits in register locations 125A and 125B set to a logic '0') then destination register 120N has invalid data, and data validity circuit 135 indicates the validity of the data in the destination register 120N by setting validity bit in register location 125N of the destination register 120N to a logic '0'. Therefore, if scheduler 105 schedules an instruction (e.g., an add instruction) in execution unit 115 and invalid data for the add instruction is loaded in either source register 120A or 120B, then after the execution unit 115 performs the add instruction, invalid data is written by the execution unit to destination register 120N. Either prior to or concurrently with execution unit 115 performing the add instruction, the data validity circuit 135 sets validity bit 125N to a logic '0' (indicating invalid data in destination register 120N).

[0024] Although, the embodiment of Figure 2 illustrates an AND gate as the data validity circuit 135, other embodiments may use other circuits and/or gates for the data validity circuit. For example, if a cache miss occurs and the validity bit in the source register is set to a logic '0', an OR gate may be used to detect the validity of the data in the source registers. The OR gate writes a logic 0 to the destination register (indicating the data in the destination register is invalid) if either or both of the source registers have invalid data.

[0025] Figure 3 illustrates a flow diagram for processing instructions according to one embodiment of the invention. As illustrated in Figure 3, at 305 for a particular instruction executed by processor 77 a determination is made whether the validity bit is set (e.g., to a logic '1') in a source e.g., a source register (indicating invalid data). If the validity bit is not set in the source register, the instruction that is executed by execution

unit 115 is retired (see Figure 1). If the validity bit is set, at 310 the data validity circuit 135 sets the validity bit in the destination register. Concurrently, with setting the validity bit in the destination register, at 315 a signal is sent e.g., by the checker unit 145 to re-execute the instruction. At 320, the scheduler re-executes the instruction. After re-execution of the instruction, at 330 a determination is made whether the re-execution of the instruction yielded valid data. If during the re-execution of the instruction the destination register receives valid data, at 325 the checker unit may send a signal to scheduler 105 to remove the instruction from the scheduler, and retires the instruction that executed correctly. If the instruction did not execute properly, the execution unit re-executes the instruction.

[0026] Embodiments of the invention may be represented as a software product stored on a machine-accessible medium (also referred to as a computer-accessible medium or a processor-accessible medium). The machine-accessible medium may be any type of magnetic, optical, or electrical storage medium including a diskette, CD-ROM, memory device (volatile or non-volatile), or similar storage mechanism. The machine-accessible medium may contain various sets of instructions, code sequences, configuration information, or other data to execute the method illustrated in Figure 3.

[0027] Thus, a method and apparatus have been disclosed for executing instructions in a processor. While there has been illustrated and described what are presently considered to be example embodiments of the present invention, it will be understood by those skilled in the art that various other modifications may be made, and equivalents may be substituted, without departing from the true scope of the invention. Additionally, many modifications may be made to adapt a particular situation to the teachings of the present invention without departing from the central inventive concept described herein. Therefore, it is intended that the present invention not be limited to

the particular embodiments disclosed, but that the invention include all embodiments falling within the scope of the appended claims.